Somewhere2 – A Robust Package for Collaborative Decentralized Consequence-Finding

Philippe Chatalic, André de Amorim Fonseca

▶ **To cite this version:**

Philippe Chatalic, André de Amorim Fonseca. Somewhere2 – A Robust Package for Collaborative Decentralized Consequence-Finding. 7th International Symposium on Intelligent Distributed Computing, Sep 2013, Prague, Czech Republic. pp.103–108, 10.1007/978-3-319-01571-2_13 . hal-01139231

HAL Id: hal-01139231
https://hal.science/hal-01139231

Submitted on 3 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

SOMEWHERE2 - A robust package for collaborative decentralized consequence-finding

Philippe Chatalic and Andre de Amorim Fonseca

Abstract This paper presents SOMEWHERE2, a new framework that may be used for setting up peer-to-peer inference systems and for solving consequence finding problems in a completely decentralized way. It is a complete redesign and reengineering of an earlier platform. The new architecture has gained in genericity, modularity and robustness. It is much easier to extend and/or to reuse as a building block for advanced distributed applications, such as Peer Data Management Systems.

1 Introduction

The *consequence finding* problem [9, 10] amounts to finding formulas that are consequences of a logical theory. Many applications involve reasoning tasks that aim at discovering such consequences, not explicit in the original theory. Often, not all consequences are sought, but only a subset of those, satisfying some syntactical property, called a *production field* [12]. Consequence finding is more complex than the *proof finding* problem, for which a user simply wants to *verify* whether a formula is entailed or not by a theory. It has proved to be useful for wide range of problems involving diagnosis, abductive reasoning, hypothetical and non-monotonic reasoning, query rewriting as well as knowledge compilation (see [10] for a survey).

There are several reasons to consider this problem in a distributed setting. For large theories, the problem may rapidly become out of scope for a single computing unit. Exploiting structural properties of the original theory in order to decompose it into subparts is one possible approach. It has been explored in the context of theorem proving by [2] and recently extended to the case of consequence finding in [3]. But the need for a distributed approach becomes essential when the knowledge is

Philippe Chatalic
L.R.I. - Bat 650, Université Paris-Sud, Orsay, France. e-mail: chatalic@lri.fr

Andre de Amorim Fonseca
L.R.I. , Inria Saclay Ile-de-France, Orsay, France. e-mail: andre.amorimfonseca@gmail.fr

intrinsically scattered at different places. This is the case in some multi agent architectures, where each agent is not necessarily willing (e.g. for some privacy reasons) to share all of its knowledge, but has to collaborate with others in order to achieve its goals. Similarly, semantic data management systems exploit the content of multiple sources of information, each of them beeing described using its own ontology. Query answering over such networked data sources requires reasoning over distributed ontologies. It generally proceeds in two steps, the first of which is a query rewriting step (that can be reformulated as a consequence finding problem), where the original query is rewritten in terms of the languages of the different relevant ontologies. Obtained rewritings are then evaluated on the appropriate sources.

Given the ever growing number of information sources available over the web, peer-to-peer (P2P) architectures look particularly promising for that purpose. The absence of any centralized control or hierarchical organization and the fact that each peer plays the same role gives much flexibility to accommodate to the dynamic nature of such networks. This also contributes to the scalability and the robustness of such approaches. Such principles are at the core of Peer Data Management Systems (PDMS) such as as EDUTELLA [11], PIAZZA [5] or SOMEWHERE [1].

SOMEWHERE is a framework based on a decentralized propositional P2P inference system (P2PIS) that can be used as a corner stone for designing elaborated PDMS. Its scalability on fairly large networks of peers has been very encouraging. It is however rather a *proof of concept* than a rock solid piece of code. Unstable, it missed essential features, e.g. the ability to cope with the dynamicity of the network. Moreover, costs for its maintenance and attempts to add new features turned out to be extremely high. At some time, the best solution has appeared to start a complete reengineering, in order to improve both its design, robustness and extensibility. The main contribution of this paper is to present the core architecture of this new system.

2 Consequence Finding in P2P Inference Systems

SOMEWHERE is based on a decentralized consequence finder that consider P2PIS $\mathscr{P} = \{P_i\}_{i=1..n}$ such as the one of Fig. 1, where each peer has its own *vocabulary* V_i (a set of propositional variables) and a local clausal theory $P_i = O_i \cup M_i$. O_i denotes the set of *local* clauses, that are made exclusively of literals over V_i (here symbols indiced by i), while M_i denotes *mapping* clauses, involving the vocabulary of at least two different peers. Intuitively, local clauses describe the very own knowledge of the peer P_i while mappings state logical constraints between different peer theories. Variables appearing in several peers are said to be *shared* (edges labels on fig. 1). They characterize possible interactions between peers and implicitly define an *acquaintance graph*. We assume each peer to be aware of its acquaintances and denote by $\text{ACQ}(l, P_i)$ the set of peers with which P_i shares the variable of a literal l. The *global theory* $P = \bigcup_{i=1..n} P_i$ is a set of clauses over the vocabulary $V = \cup_{i=1..n} V_i$.

For such networks, we consider the classical semantics of propositional logic. We use $\models$ to denote the classical consequence relation. A clause c is an *implicate* of

a theory Σ iff $\Sigma \models c$. An implicate c is *prime* iff for any other implicate c" of Σ, c" $\models c$ implies c" $\equiv c'$. By extention a clause c' is said to be a *(prime) implicate of* a clause c *wrt* Σ iff it is a (prime) implicate of $\Sigma \cup \{c\}$. Furthermore, c' is said to be a *proper* (prime) implicate of c wrt Σ if $\Sigma \cup \{c\} \models c'$, but $P \not\models c'$.

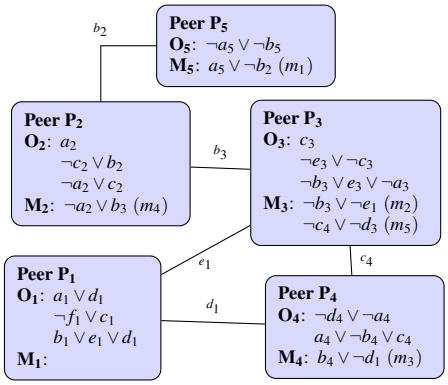

Fig. 1 A P2PIS network

Decentralized consequence finding Given a P2PIS $\mathscr{P}$, the problem we address is to compute all the *proper prime implicates* of a clause c with respect the global theory P. The point is that while c is stated using the language $\mathscr{L}_{V_i}$ (clauses over V_i) of the queried peer, the proper prime implicates of c can be clauses of $\mathscr{L}_V$[1]. Moreover, none of the peer in the P2PIS has a global view of the network. A peer only knows is its own theory P_i and the variables shared with its neighbours.

DECA [1] is the first sound and complete decentralized algorithm that has been proposed to solve this problem. It proceeds using a split/recombination strategy. When a peer P_i is asked to compute the proper prime implicates of a literal q, it first computes the proper prime implicates of q w.r.t. the local theory P_i. Each implicate c is then split in two subclauses $L(c)$ and $S(c)$, corresponding to the non-shared and shared literals of c. If non empty, $S(c)$ is then split in turn and for each shared literal l of $S(c)$ DECA asks its relevant neighbours $\text{ACQ}(l, P_i)$ (which are running the very same algorithm) to compute similarly the proper consequences of l wrt P. Answers of respective calls on neighbours are then recombined incrementally with $L(c)$.

Illustrative example The reader is referred to [1] for the full details on the algorithm. But to get an intuition of the work performed by DECA, let us illustrate the reasoning triggered by the query $q_0 = d_4@P_4$ that asks P_4 to compute the proper prime implicates of d_4 wrt P, on the P2PIS of Fig. 1. First, local consequents of d_4 on P_4 are computed, which gives : $\{d_4, \neg a_4, \neg b_4 \vee c_4, \neg d_1 \vee c_4\}$. But c_4 and d_1 being shared variables, this triggers two recursive queries, $q_1 = c_4@P_3$ and $q_2 = \neg d_1@P_1$.
$\mathbf{q_1}$: local consequents of c_4 on P_3 are $\{c_4, \neg d_3\}$. Since d_3 is not shared, the reasoning halts. Both clauses are returned to P_4, as answers for the query q_1.
$\mathbf{q_2}$: local consequents of $\neg d_1$ on P_1 are $\{\neg d_1, a_1, b_1 \vee e_1\}$. But e_1 being shared with P_3, this triggers a new query $q_3 = e_1@P_3$.
$\mathbf{q_3}$: local consequents of e_1 on P_3 are $\{e_1, \neg b_3\}$. But b_3 being shared with P_2, this triggers a new query $q_4 = b_2@P_2$.
$\mathbf{q_4}$: local consequents of $\neg b_3$ on P_2 are $\{\square\}$ (i.e. the empty clause, which subsumes all other consequents). $\square$ is then returned to P_3 as an answer for q_4, where it subsumes e_1 and $\neg b_3$. $\square$ is then returned to P_1, as the only answer for the query q_3.

[1] A variant problem is to focus on the proper prime implicates that belong to some production field $PF \subseteq \mathscr{L}_V$, supposed to characterize interesting clauses for the application domain.

On $\mathbf{P_1}$: $\square$ (which subsumes e_1) is recombined with b_1 and the set of consequents of $\neg d_1$ is now $\{\neg d_1, a_1, b_1\}$. This set is returned to P_4 as the answer for the query q_2.
On $\mathbf{P_4}$: these clauses are recombined in place of d_1 with the answers obtained for c_4, producing : $\{\neg d_1 \vee c_4, a_1 \vee c_4, b_1 \vee c_4, \neg d_1 \vee \neg d_3, a_1 \vee \neg d_3, b_1 \vee \neg d_3\}$. These answers are added to those previously obtained, namely $\{d_4, \neg a_4, \neg b_4 \vee c_4, \neg b_4 \vee \neg d_3\}$.

While [1] assume the global theory to be consistent, [4] has adapted this approach to the case where the local theories P_i are consistent, but not necessarily the whole theory P. Two algorithms are described : P2P-NG and WF-DECA, that can respectively detect all causes of inconsistencies and ensure that only so-called *well founded* consequents are produced. Algorithms DECA, P2P-NGand WF-DECA have many similarities but also differences. Their respective codes have been developed by different persons, at different periods of time, with different coding practices and style. Moreover, the original code from which they evolved suffered from many flaws, with lots of duplicated code and serious *cross cutting concerns* that strongly affected its modularity. Prohibitive maintenance costs and difficulties to add new functionalities have motivated a complete reengineering of the whole.

3 Architecture of SOMEWHERE2

SOMEWHERE2's design has been driven by several goals, among which the obtention of more robust an flexible code, developed according better software engineering practices. Robustness has been improved through a careful analysis of the different parts of the code in order to reduce dependencies as much as possible. This has lead to the design of several *components* corresponding to central concepts.

Flexibility has been improved by structuring the code in terms of an abstract notion of *module*. Each module addresses a specific concern. Some *required modules* are always loaded by the application. Others may be included (or not) at build time, according to the user's needs. A *module manager* is responsible for loading the appropriate modules. Essential functionalities of the various components are modeled in *abstract modules* and implemented *in concrete modules*. This module based approach greatly facilitates alternative concrete implementations of functionalities, the selection different sets of features and/or the creation of new extensions.

As seen in the illustrative example, DECA requires a local consequence finder, some way to interact with other peers, to recombine the results obtained from distant peers, to interact with the user (for asking queries, updating/adding/removing peers,..). P2P-NG and WF-DECA have similar needs, although declined in different ways. The current architecture of SOMEWHERE2 (Fig. 2) has 4 components (*Module Manager*, *User interface*, *Transport* and *Distributed Reasoning*), each of which with several modules. SOMEWHERE2's default configuration also rely on the component (*IASI Libs*), that offers reasoning services, but in centralized setting. As it can be used independently of *Somewhere2* it is seen as an external dependency.

Components have very few dependencies, represented by top-bottom adjacencies, e.g. *Distributed Reasoning* only depends on *Transport* and *IASI Libs*. Each

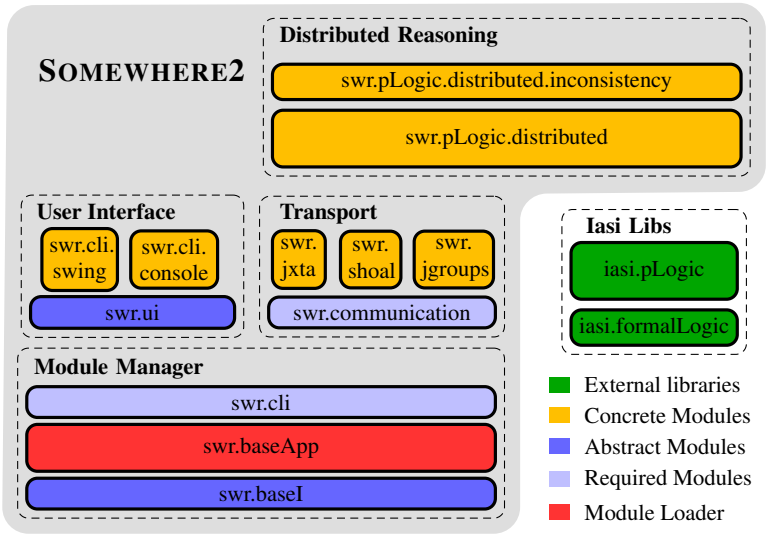

Fig. 2 SOMEWHERE2 Architectural Schema.

component can contain required, abstract, concrete and/or external modules. Module dependencies inside a component are reflected in the same way. The `baseApp` module plays a central role and is responsible for the instantiation and configuration of the other modules. Each module can have specific libraries and its own configuration scheme. We briefly describes noticeable features of some of these components.

Transport In [1], all communications were handled in an ad'hoc way, at socket level. In contrast, the new architecture is designed to reuse an existing P2P framework. The abstract module *communication* describes the concurrency model (based on http-like sessions) and core concepts required by the application for exchanges between peers (messages types, processors, ...) and the dynamicity of the network (joining/leaving peers, lost connections,...). The clean separation of abstract and concrete layers has greatly facilitated the comparison of alternative P2P frameworks (`jxta`[13], `shoal`[6] and `jgroups`[7]), without affecting other parts of the code.

Distributed Reasoning This component is responsible for all knowledge level concepts relevant to the distributed aspects of consequent finding algorithms [1, 4] (e.g. messages, handlers, network/peers modifications, anytime recombination,...). One module handle all aspects related to inconsistency tolerance and the implementations of P2P-NG and WF-DECA. Both share as much code as possible with DECA, which is implemented in the *pLogic.distributed* module.

IASI Libs Although packaged as a independent project, this component has been developed simultaneously to the other modules. It is the core library for local CF algorihms. In contrast, with the [1], that used a simple split/backward chaining strategy, SOMEWHERE2 uses a corrected and optimized version of IPIA [8].

The increased robustness of this new framework also results from a permanent effort to follow good software engineering practices, such as the systematic use of unit tests, the intensive use of design patterns and of static code analyzers (Sonar). A Jenkins server as also been configured to set up integration tests. In its current state, the project represent around 13000 lines of Java code, with less than 5% code redundancy, structured as a set of Maven projects, to ease the build process.

4 Conclusion

We have presented the architecture of SOMEWHERE2. It reunifies in a single and coherent framework two variants, tolerant or not to inconsistent theories of decentralized consequence finding algorithms. This new framework, the code of which has been has been completely rewritten and reorganized, has gained in modularity, flexibility and robustness. One noticeable improvement is the ability to deal safely with dynamic networks, with peers joining and leaving the network anytime. We expect SOMEWHERE2 to be much easier to use, to maintain and to extend. An extensive experimental study is underway and we plan to release its code under an open source licence.

References

1. P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *JAIR*, 25, January 2006.
2. E. Amir and S. McIlraith. Partition-based logical reasoning. In *KR*, pages 389–400, 2000.
3. G. Bourgne and K. Inoue. Partition-based consequence finding. In *ICTAI*, pages 641–648, 2011.
4. P. Chatalic, G.H. Nguyen, and M.C. Rousset. Reasoning with Inconsistencies in Propositional Peer-to-Peer Inference Systems. . In *ECAI*, pages 352–357, August 2006.
5. Alon Y. Halevy, Zachary Ives, Igor Tatarinov, and Peter Mork. Piazza: data management infrastructure for semantic web applications. pages 556–567. ACM Press, 2003.
6. http://shoal.java.net. Shoal – a dynamic clustering framework.
7. http://www.jgroups.org. Jgroups - a toolkit for reliable multicast communication.
8. Alex Kean and George K. Tsiknis. An incremental method for generating prime implicants/impicates. *J. Symb. Comput.*, 9(2):185–206, 1990.
9. C.T. Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms*. PhD thesis, Univ. of California, Berkeley, CA, 1967.
10. P. Marquis. *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, volume 5, chapter Consequence Finding Algorithms, pages 41–145. Kluwer Academic, 2000.
11. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, and al. Edutella: a p2p networking infrastructure based on rdf. pages 604–615. ACM, May 2002.
12. P. Siegel. *Représentation et utilisation de la connaissance en calcul propositionnel*. PhD thesis, Université d'Aix-Marseille II, 1987.
13. http://jxta.kenai.com. Jxta: A language and platform independent protocol for p2p networking.